Mastering Complexity in Formal Analysis of Complex Systems: Some Issues and Strategies Applied to Intelligent Transport Systems

Fabrice KORDON
Université Pierre & Marie Curie, Laboratoire d'Informatique de Paris 6/MoVe
4, place Jussieu, F-75252 Paris CEDEX 05, France
fabrice.kordon@lip6.fr

Abstract

Modern Intelligent Transport Systems are large, distributed, and at least partially embedded systems. They raise new challenges through safe design because of their characteristics that are not easily managed in formal methods.

The purpose of this paper is to set up a methodology that selects appropriate techniques for the modeling and analysis of such systems. Our methodology relies on Symmetric Nets (formerly known as Well Formed Petri Nets). We make intensive use of this formalism's capabilities to scale up analysis and set up a roadmap for the design of dedicated model checkers.

1 Introduction

Future systems tend to be distributed and at least partially embedded. Distribution brings a huge complexity and a strong need to deduce possible (good and bad) behaviors on the global system, from the known behavior of its actors. When such systems are embedded, new constraints of time and space may also occur as well as a strong relationship with more physical constraints (such as measures provided by DSP on the environment).

For such systems, we know that classical development methods are not adequate since the coverage of possible executions is too low [21]. This is an old observation that leads people to investigate the use of formal methods. However, these still lack in user friendly languages and tools that can enable their use by non-specialists. So, if major actors in companies or institutions dealing with critical applications acknowledge that formal methods are necessary, they also agree on the fact they must be able to scale up: today, only parts of systems are formally analyzed.

So far, there are two types of formal methods: *algebraic* approaches and *model checking*. Algebraic approaches such as B [8] allow to describe a system using axioms and then to prove a property on the specification as a theorem to be demonstrated from these axioms. These methods are very interested because the proof is parameterized. However, theorem provers that are required to elaborate the proof are difficult to use and still require highly skilled and experienced engineers.

In contrast, model checking [17] is the exhaustive investigation of a system's state space and can be automated very easily. This technique is theoretically limited by the combinatorial explosion and can mainly address finite systems. However, recent techniques based on so called symbolic techniques[1] allow to scale up to more complex systems.

So, if formal verification techniques are getting more mature, our capability to build even more complex systems also grows quickly. To catch up with problems' complexity and get fair results with formal analysis, we must fight the complexity at every stage of the process: from specification to verification itself. This requires a methodology that often make a *pragmatic* use of formal methods. By "pragmatic", we mean that assumptions are made to simplify the system. Usually, such assumptions are domain specific. So, counter to the current trend that aims to unify process development, some variations must be investigated.

This paper proposes to summarize the design methodology and techniques we use to handle very large systems throughout the modeling and verification process. We will illustrate these techniques in the context of Intelligent Transport Systems (ITS) or, in other words, mechanisms that provide driving assistance to a vehicle. This application domain is very representative of tomorrow's distributed systems where traditional programming approaches must be adapted to provide the required security. The techniques presented in this paper correspond to years of modeling and verification experience on large systems.

[1]The word *symbolic* is associated with two different techniques. The first one is based on state space encoding and was introduced in [13]. The second one relies on set-based representations of states having similar structures and was introduced in [14].

The paper is structured as follow. Section 2 presents the problems of ITS and the associated problems for verification. Then, section 3 describes the formal notation selected to model such systems. We elaborate a design methodology in section 4 and show how such specifications can be verified using appropriate model checkers (section 5).

2 Intelligent Transport Systems

In Intelligent Transport Systems (ITS), road operators, the infrastructure, vehicles, their drivers and other road users must cooperate to provide an efficient and secure system. Such systems are even more complex to analyze than previous distributed systems and require more reliability since lives can be lost. Development of such systems is a challenge supported by string research programs in Europe, USA and Japan [10].

In this section, we first illustrate some ITS issues by means of a simple example and then, discuss the major problems raised by formal modeling of such systems.

2.1 ITS Example: Safe Insertion in a Motorway

We provide a typical example for a "black-spot" (a dangerous section in the motorway). It is a freeway entrance in which we want to preserve a "Safe Insertion". Figure 1 considers a motorway with two lanes: L_1 (the rightmost one) and L_2. An entrance to the motorway, L_0, is connected to L_1. Vehicles are using the two lanes. We use the notation $V_{i,j}$, where i is the lane number and j is the vehicle identifier. $V_{0,j,}$ vehicles are entering the motorway. We want to study a cooperative insertion of vehicles arriving in the entrance lane.

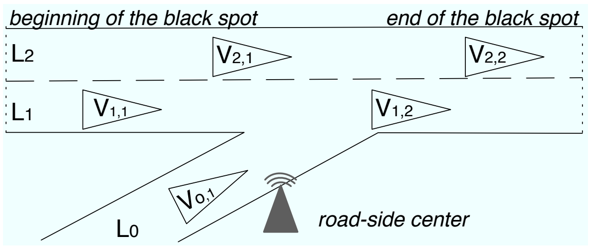


Figure 1. Safe Insertion in a motorway.

We want to let $V_{0,j}$ vehicles get into the main traffic without violating the following properties:

1. the distance between two vehicles in the same lane must be greater than a minimum safe distance to let drivers react to sudden events;
2. $V_{0,j}$ vehicles must eventually get into the motorway;
3. $V_{i,j}$ vehicles should not have to stop.

We propose to esure a safe entrance thanks to the following strategy:

(a) The motorway has a *road-side center* (RSC) that enables communication with vehicles and can compute commands related to safety or flow control.

(b) Vehicles receive their positions thanks to a satellite localization technology [11] (it may be combined with ground installations and digitized maps) and send them periodically to the infrastructure. Subsequently, the infrastructure is able to maintain a dynamic map of all vehicles in its range of communication.

(c) The infrastructure, vehicles behaviors and interactions follow an interaction cycle divided in three main steps: 1) vehicles get their positions from the satellite localization system, 2) they send this information to the infrastructure and 3) when the infrastructure has all positions of all vehicles, it issues commands according to its strategy.

Let us suppose to simplify the problem that all vehicles $V_{i,j}$ are equipped with communication devices and that the drivers follow instructions provided by the road-side center (currently, non equipped vehicles are also considered and modeled differently).

2.2 Modeling Issues for ITS

In such systems, two types of properties are of interest: *quantitative* (i.e. performance) and *qualitative* (possibility of having a wrong behavior). We focus here on qualitative properties on the system. The main goal is to understand the system's behavior in order to elaborate successful strategies. Implementation details will follow later on.

Modeling and verifying this type of system raise the following concerns:

(i) we must manage dynamic actors like cars that enter and leave the black-spot,

(ii) there are physical aspects to be modeled,

(iii) we must preserve a fair progression of the system in order to avoid having an actor perform several actions while others do nothing.

2.3 Solving ITS Modeling Issues

There are several modeling issues to solve. We first have to select an appropriate notation to model the system. We then have to elaborate a design methodology based on this notation.

It is obvious that UML is not suitable for a formal analysis of a system's behavior. It is useful to structure the system. However, the relationship between UML class, behavior and state diagrams is not yet sufficiently precise to enable formal analysis of the model's behavior. If recent evolutions of UML bring a more precise semantics, the connection between diagrams is still variously interpreted.

Algebraic techniques such as B can be useful for the verification of behavioral components as Siemens proved in the METEOR project [1]. However, it was also known as a difficult technique to automate compared to model checking based approaches. We thus selected the latter type of techniques to provide more automated tools.

Model checking could be performed on tools managing time such as UPPAAL [7] that relies on timed automata or TINA [6] that relies on Timed Petri Nets. However, we do not need explicit time management and verification of timing constraints prevents us from analyzing larger systems based on our very latest techniques.

We selected Symmetric Nets. Symmetric Nets were formerly known as Well-Formed Nets, a subclass of High-level Petri Nets. The name "Symmetric Nets" have been chosen in the context of the ISO standardization [23]. This notation provides facilities that are of interest for the analysis of complex systems if we select an appropriate modeling methodology and techniques.

In the remainder of this paper, we present the Symmetric Net formalism in section 3. Then, section 4 describes our proposal for a methodology selecting efficient modeling strategies to solve the problems mentioned in section 2.2. Section 5 finally presents how our verification techniques can scale in the description of state spaces for this type of systems.

3 Symmetric Nets, a Formal Modeling Technique

The goal of this section is to provide an informal view of Symmetric Nets. Formal definitions can be found in [14, 20].

3.1 Basics on Symmetric Nets

Let us introduce Symmetric Nets (SN) by means of a small example. The Petri net in figure 2 represents a class of threads (identified by an identity in type *P*) accessing a critical resource *CR*. Threads can get a value within the type *Val* from **CR**. Constants **PR** and **V** are parameters for the system.

The class of threads is represented by places **out** and **compute**. The place **compute** corresponds to some computation on the basis of the value provided by **CR**. At this stage, each thread holds a value that is replaced when the

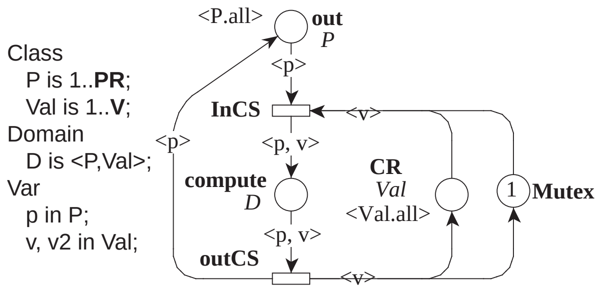


Figure 2. Example of Symmetric Net.

calculus is finished. Place **Mutex** handles mutual exclusion between threads. Place **out** initially holds one token for each value in *P* (the marking is then noted $< P.all >$) and place **CR** holds one value for each value in *Val*. Place **Mutex** only contains one token with no value.

Transitions represents evolution of the system. A transition is fired when all precondition places hold a sufficient marking. For example, Transition **inCS** can be fired if there is one token in **out**, one token in **CR** and one token in **Mutex**. When it fires, it is associated to a binding represented by the values of *p* and *v* (place **Mutex** has no type). When this transition is fired, the tuple $< p, v >$ is dropped into the postcondition place.

3.2 Correspondence with Place/Transition Nets

Simple Petri Nets are of interest since it is possible to compute structural properties. Structural properties are formulas that can be computed without exploring the full state space [20]. Here are two examples of interesting properties:

- invariants: a conservative formula on tokens in places or transitions
- bounds: the minimum and maximum number of tokens

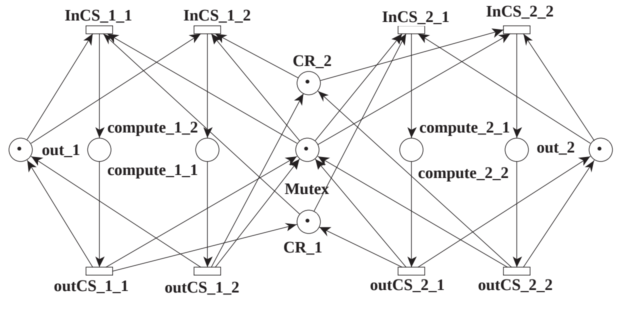


Figure 3. Unfolded P/T Net from figure 2.

It is of interest to note that such properties can be computed after an *unfolding*. This operation deploys the SN into an equivalent PTN to enable the computation of structural properties.

The principle of unfolding is simple: an SN-place is transformed into a set of PTN-places where each PTN-place represents a possible value stored in the SN-place. Let us illustrate this correspondence in Figure 3 that represents the PTN associated with the SN of figure 2 with $P = [1..2]$ and $Val = [1..2]$. Thanks to specialized decision diagram based techniques, unfolding of large models can be handled [25].

What makes this technique interesting is that it allows computing of some structural properties that are difficult to compute in SPNs. In the model of figure 2, it is obvious that the formula $card(out) + card(compute)$, where $card(p)$ represents the number of tokens in place p remains constant all over the state space. There is a projection in the unfolded model with the formula $card(card(out\_1) + card(out\_2) + compute\_1\_1) + card(compute\_1\_2) + card(compute\_2\_1) + card(compute\_2\_2)$. Section 4 provides details on how and when such properties can be used.

3.3 Symbolic Reachability Graph

Besides computation of structural properties, Petri Nets allow elaboration of the state space of the systems for model checking, thanks to the firing rule. The state space is usually called *reachability graph* and represents all concrete states of the system. Figure 4 presents the reachability graph for the Petri net of figure 2 with constants **PR** and **V** equal to 2. This state space has 5 states (the initial state is represented by a double circle) ; it will grow following the cardinality of the cartesian product $P \times Val$.

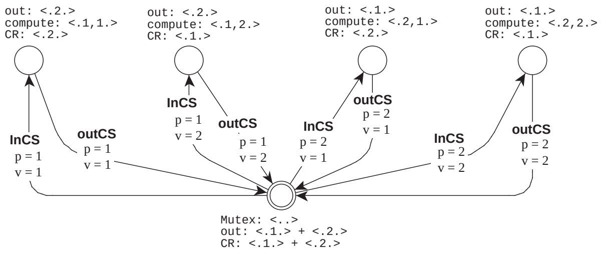


Figure 4. Reachability graph for the model of Figure 2.

The main interest of SN resides in their potential to express symbolic states in a system using the *symbolic reachability graph*. A state in the symbolic reachability graph does not represent a concrete state but a set of concrete states that have a similar structure. This is well illustrated in figure 5. The symbolic reachability graph is composed with two nodes only and will not grew when the types P and Val get more values.

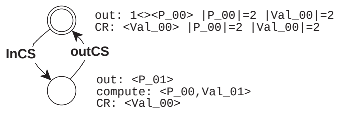

out: 1<><P_00> |P_00|=2 |Val_00|=2
CR: <Val_00> |P_00|=2 |Val_00|=2
InCS
outCS
out: <P_01>
compute: <P_00,Val_01>
CR: <Val_00>

Figure 5. Symbolic reachability graph for the model of Figure 2.

The definition of states in figure 5 must be read as follow. In the initial state, all possible values in type P are stored in place **out** and all possible values in type Val are stored in place **CR**. In the other state (when transition **InCS** fires), all possible values of type P but one are in place **out** and all possible values of type Val but one are in place **CR**. Place **compute** then contains one token composed with one value of type P (the one that is not in place **out**) and one value of type Val (the one that is not in place **CR**). Thus, this symbolic state represents all possible permutations of the couple of tokens extracted from places **out** and **CR** when firing transition **InCS**. This symbolic technique, based on the computed symmetries in a symmetric net [28], is successful when representing very large state space: it provides an exponential gain compared to the construction of concrete states [24]. This set-based representation is very efficient, especially when systems are symmetric, which is the case in numerous distributed and embedded systems. This is particularly the case in Intelligent Transport Systems since similar algorithms are supposed to be executed in each car.

4 Modeling Methodology and Techniques

This section presents our proposal for modeling and analyzing ITS-like systems. We first sketch a methodology and then put some emphasis on the modeling techniques that are optimal for this type of systems.

4.1 Design Methodology

A notation like Petri Nets should be associated with another notation to keep some structure to maintain the specification coherence with a minimum effort for the designer. We propose to have *SN-modules* that are "UML-like classes" containing sequential automata with interfaces expressed using basic Petri nets composition mechanisms: place fusion (asynchronous communication) or transition fusion (synchronous communication).

Figure 6 illustrates this design approach. The architecture of the specification is performed using SN-modules. The internals of these modules are assembled according to the places and/or transitions to be fusioned into a larger model to be analyzed. Of course, several hierarchical levels may be considered.

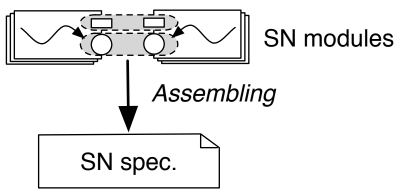


Figure 6. Overview of the design methodology.

Each SN-module may have several internals, each one corresponding to a strategy or to a given configuration to be experimented for the system. So, exploration of the system's behavior can be performed easily, similar to switching a class implementation by another one in programming languages.

To solve the specific problems identified in section 2.2, we must select appropriate modeling techniques.

4.2 Managing Dynamicity

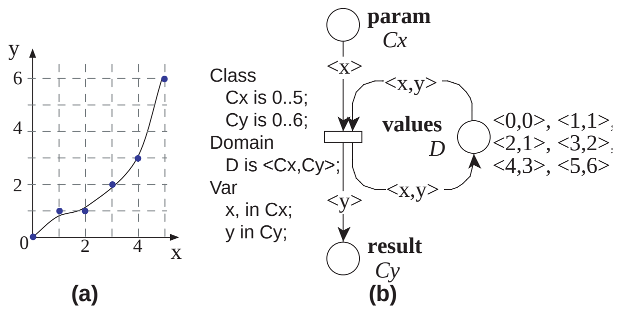
Theoretically, the number of vehicles passing into the black-spot is potentially infinite. However, we can consider that vehicles leaving the black-spot are recycled to come back into it. Thus, we can consider a finite number of vehicles according to the scenario (for example heavy traffic or light traffic). This is of particular interest since, similar to embedded systems using a thread pool, we only manage a vehicles pool.

The same technique can be used for any type of "unlimited resources". This is relevant since we do not care about resources as entities but about all possible situations they are involved in.

This technique also brings an interesting feature. The corresponding system is not expected to deadlock. Thus, a deadlock may correspond either to a property violation or to some mistake in the model itself. This is useful during the verification process.

4.3 Modeling Complex Functions

The main problem of SN is to provide only a limited set of mathematical functions to the system designer. This is required to keep the mathematical structure that enables the computation of symmetries in the specification, thus enabling the use of the symbolic reachability graph [28]. To cope with the modeling of complex functions (for example, computation of braking distance according to the current speed of a vehicle), we must discretize and represent them in a specific place. Such a place can be held in an SN-module ; it then represents the function and can be stored in a dedicated library.



```
Class
  Cx is 0..5;
  Cy is 0..6;
Domain
  D is <Cx,Cy>;
Var
  x, in Cx;
  y in Cy;
```

Figure 7. Example of complex function discretization.

Figure 7 represents an example of function discretization. Left side of the figure (a) shows a function that is discretized and the right side (b) shows the corresponding Petri net model. The function discretization is stored in place **values** ; y is obtained thanks to the unification of variable x in the two input arcs of the transition. Please note that **values**' markings remain constants.

This technique can be generalized to any function $x = f(x_1, x_2, ..., x_n)$, regardless of its complexity. Non deterministic functions can also be specified the same way (for example, to model potential errors in the system). Let us note that:

- the discretization of any function becomes a modeling hypothesis and must be validated separately (to evaluate the impact of imprecision due to discretization),
- given a programmed function, it is easy to automatically generate the list of values to store in the initial marking of the place representing the function.

The only drawback of this technique is a loss in precision compared to continuous systems that require appropriate hybrid techniques [15]. If such a discretization enables the use of more user-friendly techniques, they must be checked. For example, if we consider distances in our black-spot example, we must ensure that uncertainty remains in a safe range. This means that our metrics must be compliant with the precision to ensure, for example, that if $V_{1,1}$ follows $V_{1,2}$, the minimum distance ensures that no intersection between the associated volumes is possible.

4.4 Preserving a Fair Execution

In this type of system, all actors simultaneously behave in parallel. It is thus not reasonable to exhibit problems related to the fact that one actor progress while all others are not executed. The modeling solution is there to relate the model to a timeline that beat the execution of the system.

This timeline can be modeled explicitly or be implemented in the firing rule that includes fairness execution of the Petri Net token game.

If we consider our black-spot example, the cycle can be trivially extracted from the behavior of a vehicle as described in point (c) in section 2.1. Here the timeline ensure a sliced execution of the specification. At each time unit (not necessarily counted), all vehicle make a move and the infrastructure takes decisions to be executed during the next slice. Communication delays can also be implemented when required.

5 Towards Analysis of ITS System

Let us note that the type of system we describe here are very symmetric: vehicles can be permuted easily. Thus, the computation of symmetries that enable the use of the SN's symbolic reachability graph (see section 3.3) can be operated successfully.

Nevertheless, analysis remains quite difficult since currently implemented model checkers are not sufficient. The ones that implements a concrete state space cannot handle more than a few 10^8 states.

GreatSPN [3], a model checker implementing the symbolic reachability graph was successfully used to analyze a middleware core having about 10^{18} concrete states [24] but it seems inadequate for the complexity of ITS systems when discretization is realistic and requires types with many values (in [12], only small configurations could be analyzed). This is also observed for model checker that support a symbolic encoding of the state space such as SVM [5].

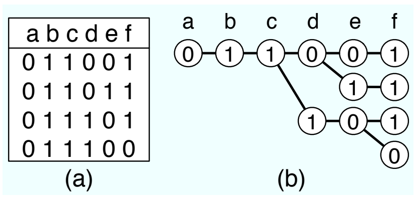
Our diagnosis, according to an analysis of the model checkers' behaviors shows that current techniques are not yet able to scale up for these systems. There is also some side effect from the modeling technique that must be considered in the model checker as domain specific optimizations.

However, we are confident that model checkers will soon be able to analyze ITS-like systems thanks to the following techniques:

- the use of symbolic/symbolic techniques,
- the design of parallel model checkers in clusters of machines,
- the management of stable marking,
- the use of hierarchical encoding techniques.

5.1 Symbolic/Symbolic Techniques

We already mentioned the symbolic reachability graph in section 3.3. Symbolic reachability graph provides similar performance in mastering the complexity of large state spaces to the encoding of states using decision diagrams [13] (also called *symbolic techniques*).



a b c d e f
0 1 1 0 0 1
0 1 1 0 1 1
0 1 1 1 0 1
0 1 1 1 0 0

Figure 8. Principles of symbolic encoding of states.

The principle of state encoding is illustrated in figure 8. Let us consider that a state in the system is represented as a Boolean vector defining the values of a set of variables *a* to *f* (part (a)). If we assume that an action in the system does not change the entire vector, we can consider a differential encoding of states. In our example, variables *a* to *c* do not change : it is unnecessary to represent their value more than once. The Binary Decision Diagram (BDD) (part b)) encodes this system and promote share of common parts in the system. The main drawback of this technique is that its efficiency is strongly related to the variable order ; if we encode the BDD from *g* to *a*, sharing performances are very poor.

This techniques was successfully elaborated to analyze hardware systems. It has been enhanced and numerous decision diagram based techniques are now available. One of these technique, Data Decision Diagrams (DDD) [18], has been elaborated to encode discrete values instead of binary ones. It is a basis to support symbolic/symbolic techniques [29]: the *symbolic encoding* of the *symbolic reachability graph*. This technique seems promising for the storage of very large state spaces.

5.2 Parallel model checking on a cluster

As shown before, the main problem of model checking is memory consumption. However, with diagram decision based techniques, another problem arises. The principle of these techniques is to trade memory against CPU. As a typical example, when a new symbolic state is computed, it has to be compared to existing ones. This requires all states to be canonized in order to have a common and comparable representation suitable for comparison. Even optimized, such an operation requires CPU.

So, distributing a model checker on a cluster of machine brings two advantages:

- states are generated in parallel thanks to an appropriate hash function,

- the model checkers takes advantage of the CPU and memory available in the whole system.

Expected results are promising and the research community is currently working in this direction. As an example, the famous SPIN model checker has already been experimented in a parallel way [26, 9]. We also successfully implemented a parallel version of GreatSPN that provides supra-linear acceleration factor for many examples in our benchmark [22].

5.3 Management of Stable Marking

The technique presented in section 4.3 generate places for which marking is large and remains constant. This should be taken into consideration by model checkers. In fact, a model checker like GreatSPN does not handle such cases and thus, this stable marking is reproduced for each generated state, thus leading to a huge memory consumption.

For model checkers using symbolic encoding of states (as well as for a symbolic/symbolic model checker), such places should be detected since their marking is highly shared by all states in the system. A pre-analysis of the specification can easily detect such configuration and provide hints for an appropriate encoding technique.

5.4 Hierarchical Encoding Techniques and Recursive Folding

Symbolic encoding of a state space (concrete or symbolic) relies on the sharing of state patterns in the state space of a system. Recent work investigates a hierarchical representation that could increase the sharing of such patterns on a larger scale. For that purpose, new representations, such as Set Decision Diagrams (SDD) [19] are being investigated.

In some favorable case (i.e. very regular symmetries in the system), the results are fantastic. Let us analyze what can be provided for the dining philosopher problem [2], as experienced in [27].

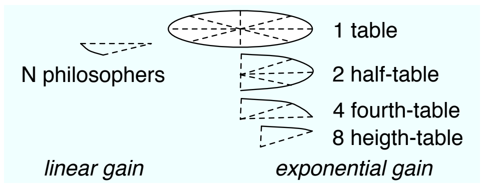


Figure 9. Possible structuring of the dining philosopher problem.

Figure 9 shows the structure of the table in the dining philosopher. For *N* philosophers, the symbolic reachability graph provides a linear factor: the philosopher pattern is automatically extracted and the gain is thus linear, leading to the analysis of just a few 10^3 philosophers, like in [16] (left side of the figure). Based on SDD, a recursive folding of the problem can be easily encoded. Instead of repeating a pattern N times, we consider a philosopher table as being two 1/2 tables or four 1/4 tables, etc. (right side of the Figure). Using this recursive encoding technique, we were able to store the state space for to 2^{10000} in a 512 Mbyte machine.

Generalization of such a technique is still a challenge but since peer-to-peer approaches, like in ITS, are usually very symmetric (as for the philosopher problem), we are confident that, in some cases, such a representation is possible.

6 Conclusion

In this paper, we have summarized the design methodologies and techniques we have developed to model and analyze very large systems. We are currently working on problems similar to the ITS case study that was presented as an illustration of future complex systems to be designed, analyzed and implemented.

To scale up in the formal analysis of such complex systems, we must work "vertically". It means that all phase of the modeling and analysis process must cooperate. This can be done in two ways:

- Some model checking techniques can be stacked to handle larger systems, as in symbolic/symbolic model checking techniques.

- When possible, a modeling technique should be coupled with the corresponding model checking technique that enables reduction of the state space. For example, the discretization of complex functions by means of a SN-place having a stable marking must be considering to encode a state by means of decision diagram to represent the marking of this place once.

Our methodology and techniques should provide better facilities to engineers who have to cope with such problems. We use these advanced model checking techniques and emphasize domain specific optimizations in associated tools. Most of the techniques presented in this paper have been implemented, either as prototypes or in CPN-AMI [4]. The objective is to provide a prototype for a CASE environment dedicated to the design and analysis of complex systems that includes all these techniques in one single model checker.

Later on, we can imagine that model checkers will be elaborated *on the fly*, according to the techniques enabled to analyze a specification according to domain specific characteristics.

References

[1] Atelier B, `http://www.atelierb.societe.com/index_uk.htm`, 2006.

[2] Dining philosophers problem, `http://en.wikipedia.org/wiki/Dining_philosophers_problem`, 2006.

[3] GreatSPN V2.0, `http://www.di.unito.it/~greatspn/index.html`, 2006.

[4] The CPN-AMI Home page, url : `http://www.lip6.fr/cpn-ami`, 2006.

[5] The SMV System, `http://www.cs.cmu.edu/~modelcheck/smv.html`, 2006.

[6] TINA, TIme petri Net Analyzer, `http://www.laas.fr/tina`, 2006.

[7] UPPAAL home page, `http://www.uppaal.com/`, 2006.

[8] J.-R. Abrial. *The B book - Assigning Programs to meanings*. Cambridge University Press, 1996.

[9] J. Barnat, V. Forejt, M. Leucker, and M. Weber. DivSPIN - a SPIN compatible distributed model checker. In M. Leucker and J. van de Pol, editors, *4th International Workshop on Parallel and Distributed Methods in verifiCation (PDMC'05)*, Lisbon, Portuga, 2005.

[10] R. Bishop. Intelligent Vehicle R&D: a review and contrast of programs worldwide and emerging trends. In J. Ehrlich, editor, *Annals of Telecommunications - Intelligent Transportation Systems*, volume 60, pages 228–263. GET-Lavoisier, March-April 2005.

[11] J.-M. Blosseville. Driving assistance systems and road safety: State-of-the-art and outlook. In J. Ehrlich, editor, *Annals of Telecommunications - Intelligent Transportation Systems*, volume 60, pages 281–298. GET-Lavoisier, March-April 2005.

[12] F. Bonnefoi, L. Hillah, F. Kordon, and G. Frémont. An approach to model variations of a scenario: Application to Intelligent Transport Systems. In *Workshop on Modelling of Objects, Components, and Agents (MOCA'06)*, Turku, Finland, June 2006.

[13] J. Burch, E. Clarke, and K. McMillan. Symbolic model checking: 10^{20} states and beyond. *Information and Computation (Special issue for best papers from LICS90)*, 98(2):153–181, 1992.

[14] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On well-formed coloured nets and their symbolic reachability graph. In K. Jensen and G. Rozenberg, editors, *Procedings of the 11th International Conference on Application and Theory of Petri Nets (ICATPN'90). Reprinted in High-Level Petri Nets, Theory and Application.* Springer-Verlag, 1991.

[15] P. Christofides and N. El-Farra. *Control Nonlinear And Hybrid Process Systems: Designs for Uncertainty, Constraints And Time-delays*. SPringer Verlag, 2005.

[16] G. Ciardo, G. Luettgen, and R. Siminiceanu. Efficient symbolic state-space construction for asynchronous systems. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets*, volume 1825 of *Lecture Notes in Computer Science*, pages 103–122. Springer-Verlag, 2000.

[17] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[18] J.-M. Couvreur, E. Encrenaz, E. Paviot-Adet, D. Poitrenaud, and P.-A. Wacrenier. Data decision diagrams for Petri net analysis. In *Proc. of ICATPN'2002*, volume 2360 of *Lecture Notes in Computer Science*, pages 101–120. Springer Verlag, June 2002.

[19] J.-M. Couvreur and Y. Thierry-Mieg. Hierarchical decision diagrams to exploit model structure. In *25th International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'05)*. Springer Verlag, to be published, October 2005.

[20] C. Girault and R. Valk. *Petri Nets for Systems Engineering*. Springer Verlag - ISBN: 3-540-41217-4, 2003.

[21] J. Gogen and Luqi. Formal methods: Promises and problems. *IEEE Software*, 14(1):75–85, 1997.

[22] A. Hamez, F. Kordon, and Y. Thierry-Mieg. libDMC: a Library to Operate Efficient Distributed Model checking. Technical report, Master's thesis, LIP6, Université P. & M. Curie, 2007.

[23] L. Hillah, F. Kordon, L. Petrucci, and N. Trèves. PN standardisation : a survey. In *International Conference on Formal Methods for Networked and Distributed Systems (FORTE'06)*, pages 307–322, Paris, France, September 2006. IFIP.

[24] J. Hugues, Y. Thierry-Mieg, F. Kordon, L. Pautet, S. Baarir, and T. Vergnaud. On the Formal Verification of Middleware Behavioral Properties. In *9th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'04)*, pages 139–157. Elsevier, September 2004.

[25] F. Kordon, A. Linard, and E. Paviot-Adet. Optimized Colored Nets Unfolding. In *International Conference on Formal Methods for Networked and Distributed Systems (FORTE'06)*, pages 339–355, Paris, France, September 2006. IFIP.

[26] F. Lerda and R. Sisto. Distributed-memory model checking with SPIN. In *Proc. of the 5th International SPIN Workshop*, volume 1680 of *LNCS*. Springer-Verlag, 1999.

[27] Y. Thierry-Mieg. *Techniques for the model checking of high-level specifications*. PhD thesis, Université P. & M. Curie, 2004.

[28] Y. Thierry-Mieg, C. Dutheillet, and I. Mounier. Automatic symmetry detection in well-formed nets. In *Proc. of ICATPN 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 82–101. Springer Verlag, June 2003.

[29] Y. Thierry-Mieg, J.-M. Ilié, and D. Poitrenaud. A symbolic symbolic state space representation. In *24th International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'04)*, pages 276–291. Springer Verlag, LNCS 3235, July 2004.